# Learning to Search via Retrospective Imitation

**Jialin Song**[*]
Caltech
jssong@caltech.edu

**Ravi Lanka**[*]
JPL
ravi.kiran@jpl.nasa.gov

**Albert Zhao**
Caltech
azzhao@caltech.edu

**Yisong Yue**
Caltech
yyue@caltech.edu

**Masahiro Ono**
JPL
masahiro.ono@jpl.nasa.gov



**Abstract**

We study the problem of learning a good search policy from demonstrations for combinatorial search spaces. We propose retrospective imitation learning, which, after initial training by an expert, improves itself by learning from its own *retrospective* solutions. That is, when the policy eventually reaches a feasible solution in a search tree after making mistakes and backtracks, it retrospectively constructs an improved search trace to the solution by removing backtracks, which is then used to further train the policy. A key feature of our approach is that it can iteratively scale up, or transfer, to larger problem sizes than the initial expert demonstrations, thus dramatically expanding its applicability beyond that of conventional imitation learning. We showcase the effectiveness of our approach on two tasks: synthetic maze solving, and integer program based risk-aware path planning.

## 1 Introduction

Many challenging tasks involve traversing a combinatorial search space. Examples include branch-and-bound for constrained optimization problems [Lawler and Wood, 1966], A* search for path planning [Hart et al., 1968] and game playing, e.g. Go [Silver et al., 2016]. Since the search space often grows exponentially with problem size, one key challenge is how to prioritize traversing the search space. A conventional approach is to manually design heuristics that exploit specific structural assumptions (cf. [Gonen and Lehmann, 2000, Holmberg and Yuan, 2000]). However, this conventional approach is labor intensive and relies on human experts developing a strong understanding of the structural properties of some class of problems.

In this paper, we take a learning approach to finding an effective search heuristic. We cast the problem as policy learning for sequential decision making, where the environment is the combinatorial search problem. Viewed in this way, a seemingly natural approach to consider is reinforcement learning, where the reward comes from finding a feasible terminal state, e.g. reaching the target in A* search. However, in our problem, most terminal states are not feasible, so the reward signal is sparse; hence, we do not expect reinforcement learning approaches to be effective.

We instead build upon imitation learning [Ross and Bagnell, 2010, Ross et al., 2011, Daumé III et al., 2009, He et al., 2014], which is a promising paradigm here since an initial set of solved instances (i.e., demonstrations) can often be obtained from existing solvers, which we also call experts. However, obtaining such solved instances can be expensive, especially for large problem instances. Hence, one key challenge that we address is to avoid repeatedly querying experts during *training*.

We propose the *retrospective imitation* approach, where the policy can iteratively learn from its own mistakes without repeated expert feedback. Instead, we use a *retrospective oracle* to generate feedback by querying the environment on rolled-out search traces (e.g., which part of the trace led to a feasible terminal state) to find the shortest path in hindsight (retrospective optimal trace).

Our approach improves upon previous imitation approaches [Ross and Bagnell, 2010, Ross et al., 2011, He et al., 2014] in two aspects. First, our approach iteratively refines towards solutions that may be higher quality or easier for the policy to find than the original demonstrations. Second and more importantly, our approach can scale to larger problem instances than the original demonstrations, allowing our approach to scale up to problem sizes beyond those that solvable by the expert, dramatically extending the applicability beyond that of traditional imitation learning algorithms. We also provide a theoretical characterization for a restricted setting of the general learning problem.

We evaluate on two environments: A* search to solve mazes, and risk-aware path planning based on mixed integer linear programs (MILPs) [Schouwenaars et al., 2001, Ono and Williams, 2008]. We demonstrate that our approach improves upon prior imitation learning work [He et al., 2014] as well as commercial solvers such as Gurobi (for MILPs). We further demonstrate generalization ability by learning to solve larger problem instances than contained in the original training data.

In summary, our contributions are

- We propose retrospective imitation, a general learning framework that can generate feedback for imitation learning algorithms without repeatedly querying experts.
- We show how retrospective imitation can scale up beyond the problem size where demonstrations are avail-

[*]Equal contribution.
Under Review @ AAAI 2019

able, which significantly expands upon the capabilities of imitation learning.

- We provide theoretical insights on when retrospective imitation can provide improvements over imitation learning, such as when we can reliably scale up.
- We evaluate empirically on two combinatorial search environments and show improvements over both imitation learning baselines and off-the-shelf solvers.

## 2 Related Work

Driven by availability of demonstration data, imitation learning is an increasingly popular learning paradigm, whereby a policy is trained to mimic the decision-making of an expert or oracle [Daumé III et al., 2009, Ross and Bagnell, 2010, Ross et al., 2011, Chang et al., 2015]. Existing approaches often rely on having access to a teacher at training time to derive learning signals from. In contrast, our retrospective imitation approach can learn from its own mistakes as well as train on larger problem instances than contained in the original supervised training set.

Another popular paradigm for learning for sequential decision making is reinforcement learning (RL) [Sutton and Barto, 1998], especially with recent success of using deep learning models as policies [Lillicrap et al., 2015, Mnih et al., 2015]. One major challenge with RL is effective and stable learning when rewards are sparse, as in our setting. In contrast, the imitation learning reduction paradigm Ross et al. [2011], Chang et al. [2015] helps alleviate this problem by reducing the learning problem to cost-sensitive classification, which essentially densifies the reward signals.

Our retrospective imitation approach bears some affinity to other imitation learning approaches that aim to exceed the performance of the oracle teacher [Chang et al., 2015]. One key difference is that we are effectively using retrospective imitation as a form of transfer learning by learning to solve problem instances of increasing size.

Another paradigm for learning to optimize is to learn a policy on-the-fly by using the first few iterations of optimization for training Ipek et al. [2008], Khalil et al. [2016]. This requires dense rewards as well as stationarity of optimal behavior throughout the search. Typically, such dense rewards are surrogates of the true sparse reward. We study a complementary setting of learning off-line from demonstrations with sparse environmental rewards.

The policy class used in some of our experiments is inspired by recent work combining tree search [Kocsis and Szepesvári, 2006] with deep learning, e.g., playing Go [Silver et al., 2016]. Since our search processes are also tree structured, such policy classes might also work well.

## 3 Problem Setting & Preliminaries

**Learning a Search Policy for Combinatorial Search Problems.** Given a combinatorial search problem instance $P$, a policy $\pi$ (i.e., a search algorithm) must make a sequence of decisions to traverse a combinatorial search space to find a (good) feasible solution (e.g., a setting of integer variables in an integer program that satisfies all constraints and has good objective value). Given the current "state" $s_t$ of the problem, which contains the search history so far (e.g., a partial assignment of integer variables in an integer program), the policy chooses an action $a$ to apply to the current state $s_t$ (i.e., to extend the current partial solution) and transitions to a new state $s_{t+1}$. The search process terminates when a complete feasible solution to $P$ is found, which we also refer to as reaching a terminal state. A typical objective is to minimize search time to a terminal state. In general, the transition function is deterministic and known, but navigating a combinatorial search space to find rare terminal states solutions is challenging. Given a training set of problem instances, a learning approach trains $\pi$ to perform well on future test problem instances.

**Imitation Learning.** We build upon the imitation learning paradigm to learn a good policy. In imitation learning, there is typically an expert policy $\pi_{expert}$ that provides interactive feedback on the trained policy He et al. [2014]. The expert can be a human or an (expensive) solver. However, a human cannot always be available, or a solver can be prohibitively expensive. Our approach is based on the idea that retrospection (with query access to environment) can also generate feedback. A search trace typically has many dead ends and backtracking before finding a terminal state. Thus, more efficient search traces (i.e., feedback) can be retrospectively extracted by removing backtracking, which forms the core algorithmic innovation of our retrospective imitation approach. This idea is formalized in Section 4. Retrospective imitation also enables a form of transfer learning where our policy can be iteratively trained to solve larger problems for which the original expert (e.g., an existing MILP solver) may be ineffective and collecting training demonstrations is infeasible, for instance due to computational complexity.

## 4 Retrospective Imitation Learning

We now describe the retrospective imitation learning approach. It is a general framework that can be combined with a variety of imitation learning algorithms. For clarity of presentation, we instantiate our approach using the data aggregation algorithm (DAgger) [Ross et al., 2011, He et al., 2014] and we call the resulting algorithm Retrospective DAgger. We also include the instantiation with SMILe [Ross and Bagnell, 2010] in Appendix A. In Section 6, we empirically evaluate retrospective imitation with both DAgger and SMILe to showcase the generality of our framework.

The ultimate goal of retrospective imitation is to enable imitation learning algorithms to scale up to problems much larger than those for which we have expert demonstrations, which is a significant improvement since conventional imitation learning cannot naturally accomplish this. To accomplish this goal, we decompose our general framework into two steps. First, Algorithm 1 describes our core procedure for learning on fixed size problems with a crucial *retrospective oracle* subroutine (Algorithm 2). Finally, Algorithm 3 describes how to scale up beyond the fixed size.

We will use Figure 1 as a running example. In Figure 1, the search trace is tree-structured, where circular and diamond nodes represent intermediate and terminal states, respectively. Numbers in nodes indicate the order visited.

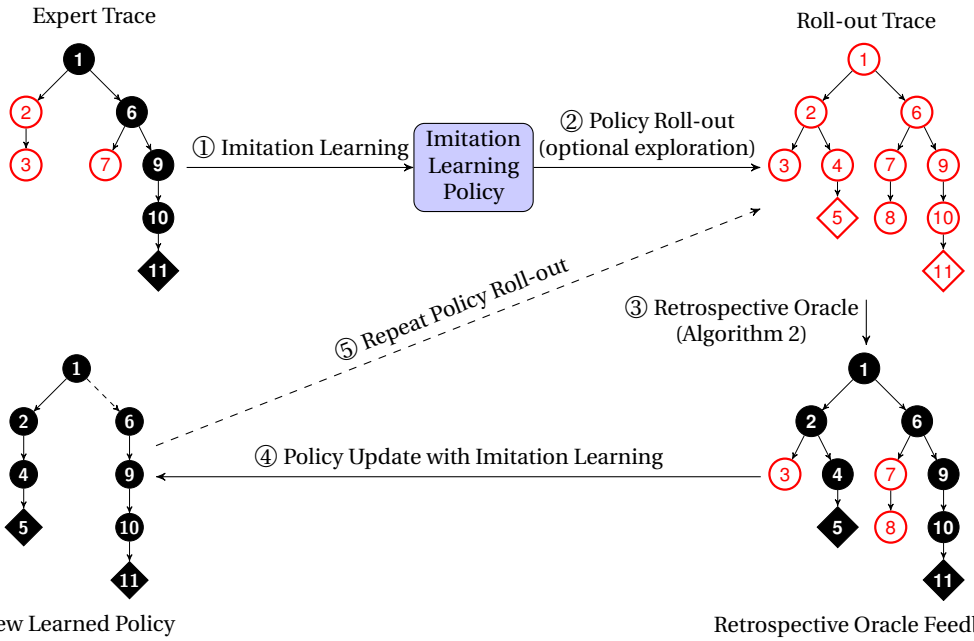



Figure 1: A visualization of retrospective imitation learning depicting components of Algorithm 1. An imitation learning policy is initialized from expert traces and is rolled out to generate its own traces. Then the policy is updated according to the feedback generated by the retrospective oracle. This process is repeated until some termination condition is met.

**Algorithm 1:** Retrospective DAgger for Fixed Size

1 **Inputs:**
2 $N$: number of iterations
3 $\pi_1$: initially trained on expert traces
4 $\alpha$: mixing parameter
5 $P$: a set of problem instances
6 $D_0$: expert traces dataset
7 $D = D_0$
8 **for** $i \leftarrow 1$ **to** $N$ **do**
9 $\quad \hat{\pi}_i \leftarrow \alpha\pi_i + (1-\alpha)\pi_{explore}$ (optionally explore)
10 $\quad$ run $\hat{\pi}_i$ on $P$ to generate trace $\tau$
11 $\quad$ compute $\pi^*(\tau, s)$ for each terminal state $s$ (Algorithm 2)
12 $\quad$ collect new dataset $D_i$ based on $\pi^*(\tau, s)$
13 $\quad$ update $D$ with $D_i$ (i.e., $D \leftarrow D \cup D_i$)
14 $\quad$ train $\pi_{i+1}$ on $D$
15 **end**
16 return best $\pi_i$ on validation

**Algorithm 2:** Retrospective Oracle for Tree Search

1 **Inputs:**
2 $\tau$: search tree trace
3 $s$: terminal state
4 **Output:**
5 retro_optimal: the retrospective optimal trace
6 **while** *$s$ is not the root* **do**
7 $\quad$ parent $\leftarrow$ $s$.parent
8 $\quad$ retro_optimal(parent) $\leftarrow$ $s$
9 $\quad$ $s \leftarrow$ parent
10 **end**
11 return retro_optimal

**Core Algorithm for Fixed Problem Size.** The core learning procedure is iterative. We assume access to an initial dataset of expert demonstrations to help bootstrap the learning process, as described in Line 3 in Algorithm 1 and depicted in step ① in Figure 1. In Lines 9-10, the current policy (potentially blended with an exploration policy) runs until a termination condition, such as reaching one or more terminal states, is met. In Figure 1, this is step ② and 2 terminal states (5 and 11) are found. In Line 11, a retrospective oracle computes retrospective optimal trace for *each* terminal state (step ③). In our example, black nodes form two retrospective optimal traces: $1 \rightarrow 2 \rightarrow 4 \rightarrow 5$ for terminal state 5 and $1 \rightarrow 6 \rightarrow 9 \rightarrow 10 \rightarrow 11$ for terminal state 11. In Line 12, a new dataset is generated as discussed below. In Lines 12-14, we imitate the retrospective optimal trace (in this case using DAgger). We then train a new policy and repeat the process.

**Retrospective Oracle.** A retrospective oracle (with query access to the environment) takes as input a search trace $\tau$ and outputs a retrospective optimal trace $\pi^*(\tau, s)$ for each terminal state $s$. Note that optimality is measured with respect to $\tau$, and not globally. That is, based on $\tau$, what is the best known action sequence to reach a terminal state if we were asked to solve the *same* instance again? In Figure 1, given the initial roll-out trace after step ② with terminal states 5 and 11, we know the optimal trace in retrospect to reach 5 is through $1 \rightarrow 2 \rightarrow 4 \rightarrow 5$. In general, $\pi^*(\tau, s)$ will be shorter than $\tau$. As we aim to minimize the number of actions taken, $\pi^*(\tau, s)$ is an effective demonstrations. Algorithm 2 shows the retrospective oracle for tree-structured search. Identifying a retrospective optimal trace given a

**Algorithm 3:** Retrospective Imitation for Scaling Up

**Inputs:**
$S_1$: initial problem size
$S_2$: target problem size
$\pi_{S_1}$: trained on expert data of problem size $S_1$
**for** $s \leftarrow S_1 + 1$ **to** $S_2$ **do**
    generate problem instances $P_s$ of size $s$
    train $\pi_s$ via Alg. 1 by running $\pi_{s-1}$ on $P_s$ to generate initial search traces
**end**

terminal state is equivalent to following parent pointers until the initial state as this results in the shortest trace.

**Design Decisions in Training Data Creation.** Algorithm 1 requires specifying how to create each new dataset $D_i$ given the search traces and a retrospective optimal trace. Intuitively $D_i$ should contain mistakes made during rollout to reach a terminal state $s$ when comparing to $\pi^*(\tau, s)$. What constitutes a mistake is influenced by the policy's actions. For example, in branch-and-bound, $D_i$'s contain data about which nodes should have been selected and pruned to reach a solution faster. Moreover, we also need to decide which terminal state(s) to prioritize in the case that multiple ones are present in $\tau$ based on their qualities. See Section 6 for concrete instantiations of these decisions for learning search heuristics for solving mazes and learning branch-and-bound heuristics for solving MILPs.

**Scaling Up.** The most significant benefit of retrospective imitation is the ability to scale up to problems of sizes beyond those in the initial dataset of expert demonstrations. Algorithm 3 describes our overall framework, whicht iteratively learns to solve increasingly larger instances using Algorithm 1 as a subroutine. We show in the theoretical analysis that, under certain assumptions, retrospective imitation is guaranteed able to scale, or transfer, to increasingly larger problem instances.

**Incorporating Exploration.** In practice, it can be beneficial to employ some exploration. Exploration is typically more useful when scaling up to larger problem instances. We discuss some exploration approaches in Appendix H.

## 5 Theoretical Results

In this section, we provide theoretical insights on when we expect retrospective imitation to improve reduction based imitation learning algorithms, such as DAgger and SMILe.

For simplicity, we regard all terminal states as equally good so we care about finding one as quickly as possible. Note that our experiments evaluate settings beyond those covered in the theoretical analysis. For brevity, all proofs are deferred to the appendix.

Our analysis builds on a trace inclusion assumption. That is, the search trace $\tau_1$ generated by a trained policy contains the trace $\tau_2$ by an expert policy. This strict assumption allows us to theoretically characterize guarantees provided by retrospective imitation. In practical experiment it may not be satisfied, but we observe that the conclusion of our theoretical result still holds up.

For performance metric, we define an error rate $\epsilon = \frac{\text{\#Non-optimal actions compared to retrospective optimal trace}}{\text{\#Actions to reach a terminal state in retrospective optimal trace}}$ to measure quality of a policy. For example, in Figure 1, the error rate along the path $1 \to 2 \to 4 \to 5$ is $\frac{1}{3}$ since there is one non-optimal move at node 2 (node 3 is explored before node 4). This leads us to the following proposition stating that retrospective imitation can effectively scale up and obtain a lower error rate.

**Proposition 1.** *Assume $\pi_{S_1}$ is a policy trained using imitation learning on problem size $S_1$. If, during the scaling-up training process to problems of size $S_2 > S_1$, the trained policy search trace, starting from $\pi_{S_1}$, always contains the expert search trace, then the final error rate $\epsilon_{S_2}$ on problems of size $S_2$ is at most that obtained by running imitation learning directly on problems of size $S_2$.*

Next we analyze how lower error rates impact the number of actions to reach a terminal state. We restrict ourselves to decision spaces of size 2: branch to one of its children or backtrack to its parent. Theorem 2 equates the number of actions to hitting time for an asymmetric random walk.

**Theorem 2.** *Let $\pi$ be a trained policy that has an error rate of $\epsilon \in (0, \frac{1}{2})$ as measured against the retrospective feedback. Let $P$ be a search problem where the optimal action sequence has length $N$. Then the expected number of actions by $\pi$ to reach a terminal state is $\frac{N}{1-2\epsilon}$.*

This result connects the error rate with our objective and implies that lower error rate leads to shorter search time in expectation. By combining this result with the lower error rate of retrospective imitation (Proposition 1), we see that retrospective imitation has a shorter expected search time than the corresponding imitation learning algorithm. We provide further analysis on this connection in the appendix.

## 6 Experimental Results

We empirically validate the generality of our retrospective imitation technique by instantiating it with two well-known imitation learning algorithms, DAgger [Ross et al., 2011] and SMILe [Ross and Bagnell, 2010]. Appendix A describes how to instantiate retrospective imitation with SMILe instead of DAgger. We showcase the scaling up ability of retrospective imitation by only using demonstrations on the smallest problem size and scaling up to larger sizes in an entirely unsupervised fashion through Algorithm 3. We experimented on two combinatorial search environments: maze solving with A* search and integer program based risk-aware path planning [Ono and Williams, 2008]. Code will be available upon publication.

### 6.1 Environments and Datasets

Both environments, A* search and integer program based risk-aware path planning, have combinatorial search spaces. The latter is particularly challenging and characterizes a practical combinatorial optimization problem. We also include additional comparisons using datasets in [He et al., 2014] in Appendix G for completeness.

**Maze Solving with A* Search.** We generate random mazes according to the Kruskal's algorithm [Kruskal, 1956].

For imitation learning, we use search traces provided by an A* search procedure equipped with the Manhattan distance heuristic as initial expert demonstrations.

We experiment on mazes of 5 increasing sizes, from 11 × 11 to 31 × 31. For each size, we use 48 randomly generated mazes for training, 2 for validation and 100 for testing. We perform A* search with Manhattan distance as the search heuristic to generate initial expert traces which are used to train imitation learning policies. The learning task is to learn a priority function to decide which locations to prioritize and show that it leads to more efficient maze solving. For our retrospective imitation policies, we only assume access to expert traces of maze size 11 × 11 and learning on subsequent sizes is carried out according to Algorithm 3. Training retrospective imitation resulted in generating ∼ 100k individual data points.

**Integer Program based Risk-aware Path Planning.** We briefly describe the risk-aware planning setup. Appendix C contains a detailed description. Given a start point, a goal point, a set of polygonal obstacles, and an upper bound of the probability of failure (risk bound), we must find a path, represented by a sequence of way points, that minimizes a cost while limiting the probability of collision to the risk bound. This task can be formulated as a MILP [Schouwenaars et al., 2001, Prékopa, 1999], which is often solved using branch-and-bound [Land and Doig, 1960]. Recently, data-driven approaches that learn branching and pruning decisions have been studied in [He et al., 2014, Alvarez et al., 2014, Khalil et al., 2016]. Solving MILPs is in general NP-hard, and this combinatorial search problem presents a practical challenge to our learning approach.

The experiment is conducted on a set of 150 different instances of randomly generated obstacle maps with 10 obstacles each. We used a commercially available MILP solver Gurobi (Version 6.5.1) to generate expert solutions. Details on dataset generation can be found in Appendix D. The risk bound was set to $\delta = 0.02$. We started from problems with 10 way points and scaled up to 14 way points, in increments of 1. The number of integer variables range from 400 to 560, which can be quite challenging to solve for the type of path planning problem of our interest.

For training, we assume that expert demonstrations by Gurobi are only available for the smallest problem size (10 way points, 400 binary variables). We use 50 instance of randomly generated obstacle maps each for training, validation and testing. Training retrospective imitation resulted in generating ∼ 1.4 million individual data points.

### 6.2 Policy Learning

For A* search, we learn a ranking model as the policy. The input features are mazes represented as a discrete-valued matrix indicating walls, passable squares, and the current location. We instantiate using neural networks with 2 convolutional layers with 32 3 × 3 filters each, 2 × 2 max pooling, and a feed-forward layer with 64 hidden units.

For risk-aware path planning, we experimented with two policy classes. The first follows [He et al., 2014], and consists of a node selection model (that prioritizes which node to consider next) and a pruning model (that rejects nodes from being branched on), which mirrors the structure of common branch-and-bound search heuristics. We use RankNet [Burges et al., 2005] as the selection model, instantiated using a 2-layer neural network, with LeakyReLU [Maas et al., 2013] activation functions and trained via cross entropy loss. For the pruning model, we train a 1-layer neural network classifier with higher cost on the optimal nodes compared to the negative nodes. We refer to this policy class as "select & pruner". The other policy class only has the node selection model and is referred to as "select only".

The features can be categorized into node-specific and tree-specific features. Node-specific features include an LP relaxation lower bound, objective value and node depth. Tree-specific features capture global aspects that include the integrality gap, number of solutions found, and global lower and upper bounds. We normalize each feature to [-1,1] across the candidate variables at each node, which is also known as query-based normalization [Qin et al., 2010].

### 6.3 Main Results

**Comparing Retrospective Imitation with Imitation Learning.** As retrospective learning is a general framework, we validate with two different baseline imitation learning algorithms, DAgger [Ross et al., 2011] and SMILe [Ross and Bagnell, 2010]. We consider two possible settings for each baseline imitation learning algorithm. The first is "Extrapolation", which is obtained by training an imitation model only using demonstrations on the smallest problem size and applying it directly to subsequent sizes without further learning. Extrapolation is the natural baseline to compare with retrospective imitation as both have access to the same demonstration dataset. The second baseline setting is "Cheating", where we provide the baseline imitation learning algorithm with expert demonstrations on the target problem size, which is significantly more than provided to retrospective imitation. Note that Cheating is not feasible in practice for settings of interest.

Our main comparison results are shown in Figure 2. We see that retrospective imitation (blue) consistently and dramatically outperforms conventional Extrapolation imitation learning (magenta) in every setting. We see in Figure 2a, 2d that retrospective imitation even outperforms Cheating imitation learning, despite having only expert demonstrations on the smallest problem size. We also note that (Retrospective) DAgger consistently outperforms (Retrospective) SMILe. We discuss these results further in the following.

In the maze setting (Figure 2a, 2d), the objective is to minimize the number of explored squares to reach the target location. Without further learning beyond the base size, Extrapolation degrades rapidly and the performance difference with retrospective imitation becomes very significant. Even compared with Cheating policies, retrospective imitation still achieves better objective values at every problem size, which demonstrates its transfer learning capability. Figure 3 depicts a visual comparison for an example maze.

In the risk-aware path planning setting (Figure 2b, 2c, 2e, 2f), the objective is to find feasible solutions with low optimality gap, defined as the percentage difference between the best objective value found and the optimal. If

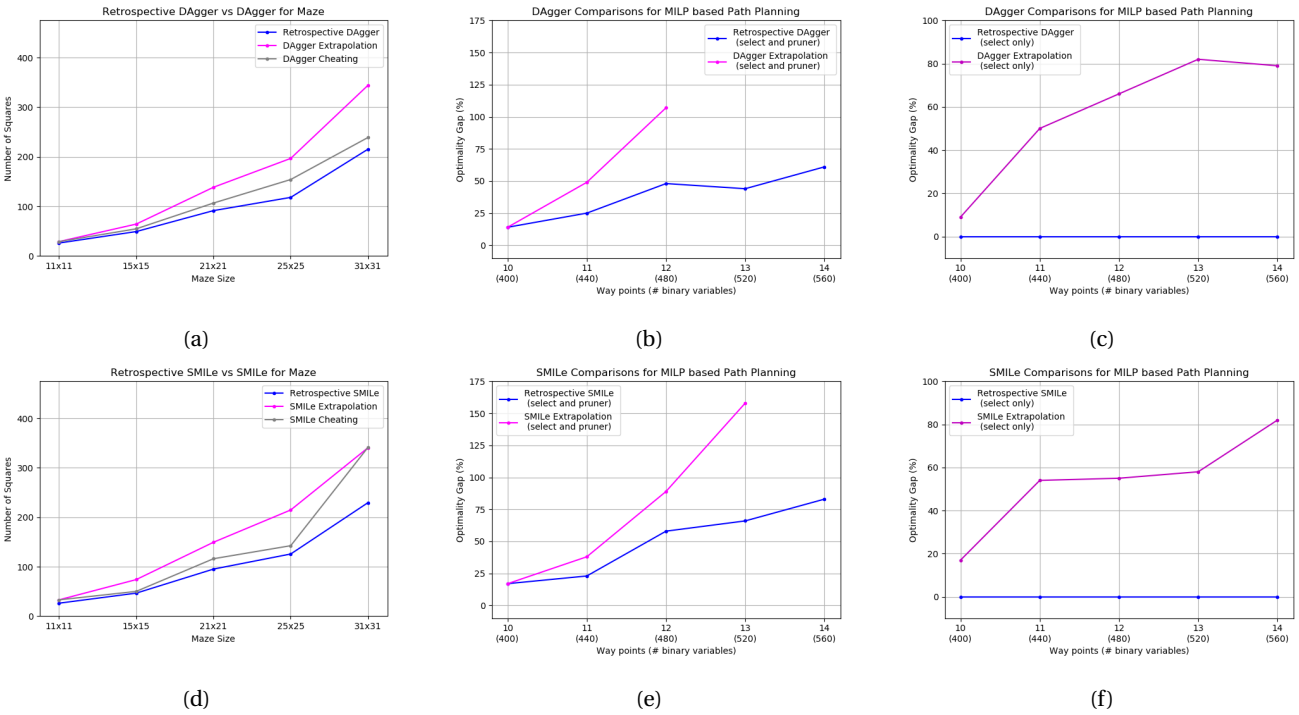



Figure 2: Retrospective imitation versus DAgger (top) and SMILe (bottom) for maze solving (left) and risk-aware path planning (middle and right, with different policy classes). "Extrapolation" is the conventional imitation learning baseline, and "Cheating" (left column only) gives imitation learning extra training data. Retrospective imitation consistently and significantly outperforms imitation learning approaches in all settings.

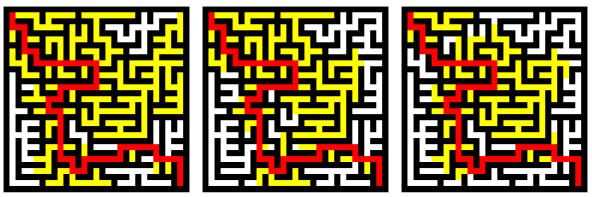


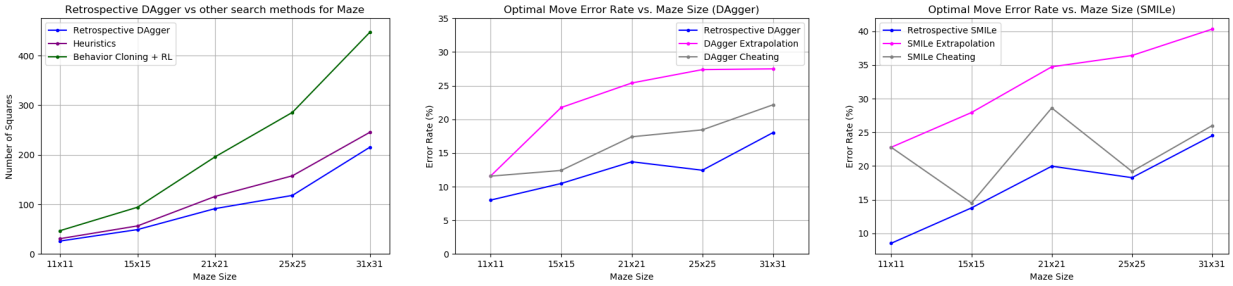
Figure 3: Left to right: comparing Manhattan distance heuristic, DAgger Cheating and Retrospective DAgger on a $31 \times 31$ maze starting at upper left and ending at lower right. Yellow squares are explored. Optimal path is red. The three algorithms explore 333, 271 and 252 squares, respectively.

a policy fails to find a feasible solution for a problem instance, we impose an optimality gap of 300%. For statistics on how many problems are not solved, see Appendix F. We compare the optimality gap of the algorithms at the same number of explored nodes. Specifically, in Figure 2b, 2e we first run the retrospective imitation version until termination, then run the other algorithm to the same number of explored nodes. In Figure 2c, 2f, we first run the retrospective imitation with the "select only" policy class until termination, then run other algorithms to the same number of explored nodes. We note that the "select only" policy class (Figure 2c, 2f) significantly outperforms the "select and pruner" policy class (Figure 2b, 2e), which suggests that utilizing conceptually simpler policy classes may be more amenable to learning-based approaches in combinatorial search problems.

While scaling up, retrospective imitation obtains consistently low optimality gaps. In contrast, DAgger Extrapolation in Figure 2b failed to find feasible solutions for $\sim 60\%$ test instances beyond 12 way points, so we did not test it beyond 12 way points. SMILe Extrapolation in Figure 2e failed for $\sim 75\%$ of the test instance beyond 13 way points. The fact that retrospective imitation continues to solve larger MILPs with a very slow optimality gap growth suggests that our approach is performing effective transfer learning.

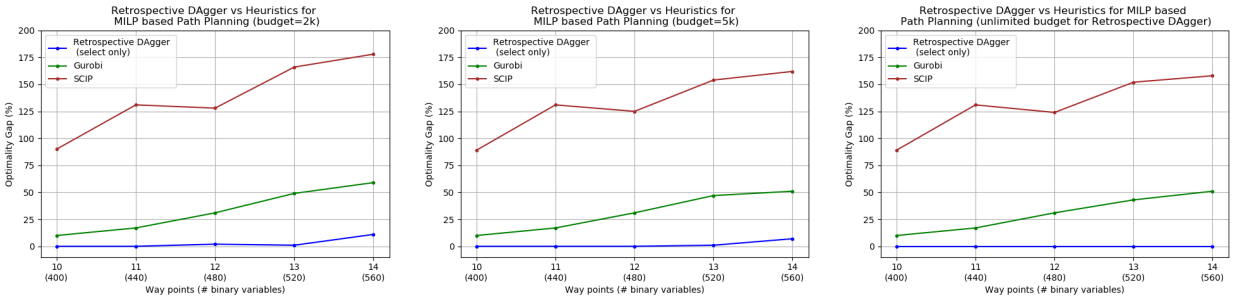
**Comparing Retrospective Imitation with Off-the-Shelf Approaches.** For maze solving, we compare Retrospective DAgger with: 1) A* search with the Manhattan distance heuristic, and 2) behavioral cloning followed by reinforcement learning with a deep Q-network [Mnih et al., 2015]. Figure 4a shows Retrospective DAgger outperforming both methods. Due to the sparsity of the environmental rewards (only positive reward at terminal state), reinforcement learning performs significantly worse than even the Manhattan distance heuristic.

For risk-aware path planning, we compare Retrospective DAgger ("select only") with a commercial solver Gurobi (Version 6.5.1) and SCIP (Version 4.0.1, using Gurobi as the LP solver). We implement our approach within the SCIP [Achterberg, 2009] integer programming framework. Due to the difference in the implementation, we use the number of explored nodes as a proxy for runtime. We control the




(a) (b) (c)

Figure 4: (left) Retrospective imitation versus off-the-shelf methods. The RL baseline performs very poorly due to sparse environmental rewards. (middle, right) Single-step decision error rates, used for empirically validating theoretical claims.




(a) (b) (c)

Figure 5: Retrospective DAgger ("select only" policy class) with off-the-shelf branch-and-bound solvers using various search node budgets. Retrospective DAgger consistently outperforms baselines.

search size for Retrospective DAgger ("select only") and use its resulting search sizes to control Gurobi and SCIP. Figure 5 shows the results on a range of search size limits. We see that Retrospective DAgger ("select only") is able to consistently achieve the lowest optimality gaps, and the optimality gap grows very slowly as the number of integer variables scale far beyond the base problem scale. As a point of comparison, the next closest solver, Gurobi, has optimality gaps $\sim 50\%$ higher than Retrospective DAgger ("select only") at 14 waypoints (560 binary variables).

**Empirically Validating Theoretical Results.** Finally, we evaluate how well our theoretical results in Section 5 characterizes experimental results. Figure 4b and 4c presents the optimal move error rates for the maze experiment, which validates Proposition 1 that retrospective imitation is guaranteed to result in a policy that has lower error rates than imitation learning. The benefit of having a lower error rate is explained by Theorem 2, which informally states that a lower error rate leads to shorter search time. This result is also verified by Figure 2a and 2d, where Retrospective DAgger/SMILe, having the lowest error rates, explores the fewest number of squares at each problem scale.

## 7 Conclusion & Future Work

We have presented the retrospective imitation approach for learning combinatorial search policies. Our approach extends conventional imitation learning, by being able to learn good policies without requiring repeated queries to an expert. A key distinguishing feature of our approach is the ability to scale to larger problem instances than contained in the original supervised training set of demonstrations. Our theoretical analysis shows that, under certain assumptions, the retrospective imitation learning scheme is provably more powerful and general than conventional imitation learning. We validated our theoretical results on a maze solving experiment and tested our approach on the problem of risk-aware path planning, where we demonstrated both performance gains over conventional imitation learning and the ability to scale up to large problem instances not tractably solvable by commercial solvers.

By removing the need for repeated expert feedback, retrospective imitation offers the potential for increased applicability over imitation learning in search settings. However, human feedback is still a valuable asset as human computation has been shown to boost performance of certain hard search problems [Le Bras et al., 2014]. It will be interesting to incorporate human computation into the retrospective imitation learning framework so that we can find a balance between manually instructing and autonomously reasoning to learn better search policies. Retrospective imitation lies in a point in the spectrum between imitation learning and reinforcement learning; we are interested in exploring other novel learning frameworks in this spectrum as well.

## References


Tobias Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1): 1–41, 2009.

Ro Marcos Alvarez, Quentin Louveaux, and Louis Wehenkel. A supervised machine learning approach to variable branching in branch-and-bound. In *European Conference on Machine Learning (ECML)*, 2014.

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *International Conference on Machine Learning (ICML)*, 2005.

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daume, and John Langford. Learning to search better than your teacher. In *International Conference on Machine Learning (ICML)*, pages 2058–2066, 2015.

Hal Daumé III, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine learning*, 75(3): 297–325, 2009.

Rica Gonen and Daniel Lehmann. Optimal solutions for multi-unit combinatorial auctions: Branch and bound heuristics. In *ACM Conference on Economics and Computation (EC)*, 2000.

Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

He He, Hal Daume III, and Jason M Eisner. Learning to search in branch and bound algorithms. In *Neural Information Processing Systems (NIPS)*, 2014.

Kaj Holmberg and Di Yuan. A lagrangian heuristic based branch-and-bound approach for the capacitated network design problem. *Operations Research*, 48(3):461–481, 2000.

Engin Ipek, Onur Mutlu, José F Martínez, and Rich Caruana. Self-optimizing memory controllers: A reinforcement learning approach. In *IEEE International Symposium on Computer Architecture (ISCA)*, 2008.

Elias Boutros Khalil, Pierre Le Bodic, Le Song, George L Nemhauser, and Bistra N Dilkina. Learning to branch in mixed integer programming. In *AAAI*, pages 724–731, 2016.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European Conference on Machine Learning (ECML)*, 2006.

Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

Ailsa H Land and Alison G Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pages 497–520, 1960.

Eugene L Lawler and David E Wood. Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719, 1966.

Ronan Le Bras, Yexiang Xue, Richard Bernstein, Carla P Gomes, and Bart Selman. A human computation framework for boosting combinatorial solvers. In *National Conference on Artificial Intelligence (AAAI)*, 2014.

Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *ACM conference on Electronic commerce*, pages 66–76, 2000.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning (ICML)*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Masahiro Ono and Brian C Williams. An efficient motion planning algorithm for stochastic dynamic systems with constraints on probability of failure. In *AAAI*, pages 1376–1382, 2008.

Masahiro Ono, Brian C Williams, and Lars Blackmore. Probabilistic planning for continuous dynamic systems under bounded risk. *Journal of Artificial Intelligence Research (JAIR)*, 46:511–577, 2013.

András Prékopa. The use of discrete moment bounds in probabilistic constrained stochastic programming models. *Annals of Operations Research*, 85:21–38, 1999.

Tao Qin, Tie-yan Liu Jun, and Xu Hang. LETOR : A Benchmark Collection for Research on Learning to Rank for Information Retrieval. *Information Retrieval*, 13(4):346–374, 2010.

Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. pages 661–668, 2010.

Stéphane Ross, Geoffrey Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

Tom Schouwenaars, Bart DeMoor, Eric Feron, and Jonathan How. Mixed integer programming for multi-vehicle path planning. In *European Control Conference*, pages 2603–2608, 2001.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

# Supplementary Material

## A Retrospective Imitation with SMILe

**Algorithm 4:** Retrospective SMILe

**Inputs:**
$N$: number of iterations
$\pi_1$: initially trained on expert traces
$\alpha$: mixing parameter
$P$: a set of problem instances
**for** $i \leftarrow 1$ **to** $N$ **do**
  run $\pi_i$ on $P$ to generate trace $\tau$
  compute $\pi^*(\tau, s)$ for each terminal state $s$ (Algorithm 2)
  collect new dataset $D$ based on $\pi^*(\tau, s)$
  train $\hat{\pi}_{i+1}$ on $D$
  $\pi_{i+1} = (1-\alpha)^i \pi_1 + \alpha \sum_{j=1}^{i} (1-\alpha)^{j-1} \hat{\pi}_j$
**end**
return $\pi_{N+1}$

## B Additional Theoretical Results and Proofs

First we prove Proposition 1.

*Proof.* By the trace inclusion assumption, the dataset obtained by retrospective imitation will contain feedback for every node in the expert trace. Furthermore, the retrospective oracle feedback corresponds to the right training objective while the dataset collected by imitation learning does not, as explained in Section 4. So the error rate trained on retrospective imitation learning data will be at most that of imitation learning. □

Our next theoretical result demonstrates that if larger problem instances have similar optimal solutions, a policy will not suffer a large increase in its error rate, i.e., we can "transfer" to larger sizes effectively. We consider the case where the problem size increase corresponds to a larger search space, i.e., the underlying problem formulation stays the same but an algorithm needs to search through a larger space. Intuitively, the following result shows that a solution from a smaller search space could already satisfy the quality constraint. Thus, a policy trained on a smaller scale can still produce satisfactory solutions to larger scale problems.

**Proposition 3.** *For a problem instance $P$, let $v_k^*$ denote the best objective value for $P$ when the search space has size $k$. Assume an algorithm returns a solution with objective value $v_k$, with $v_k \geq \alpha v_k^*$ with $\alpha \in (0,1)$. Then for any $\beta > 0$, there exists $K$ such that $v_K \geq \alpha v_{K+1}^* - \beta$.*

*Proof.* Since $P$ has a finite optimal objective value $v^*$, and for any $k < k'$, $v_k^* \leq v_{k'}^*$, then it follows that there exists an index $K$ such that $v_{K+1}^* - v_K^* \leq \frac{\beta}{\alpha}$.
Then it follows that $v_K \geq \alpha v_K^* \geq \alpha(v_{K+1}^* - \frac{\beta}{\alpha}) = \alpha v_{K+1}^* - \beta$. □

Since the slack variable $\beta$ can be made arbitrarily small, Proposition 3 implies that solutions meeting the termination condition need not look very different when transitioning from a smaller search space to a larger one. Finally, our next corollary justifies applying a learned policy to search through a larger search space while preserving performance quality, implying the ability to scale up retrospective imitation on larger problems so long as the earlier propositions are satisfied.

**Corollary 3.1.** *Let $\epsilon_k$ be the error rate of an algorithm searching through a search space of size $k$. Then there exists $K$ such that $\epsilon_K = \epsilon_{K+1}$.*

To prove Theorem 2 we need to following lemma on asymmetric 1-dimensional random walks.

**Lemma.** *Let $Z_i, i = 1, 2, \cdots$ be i.i.d. Bernoulli random variables with the distribution $Z_i = \begin{cases} 1, \text{ with probability } 1-\epsilon \\ -1, \text{ with probability } \epsilon \end{cases}$ for some $\epsilon \in [0, \frac{1}{2})$. Define $X_n = \sum_{i=1}^{n} Z_i$ and $\tau_N = \inf\{n : X_n = N\}$ for some fixed integer $N \geq 0$. Then*

*(i)* *$\{X_n + (2\epsilon - 1)n\}$ is a martingale with respect to the filtration $\{\mathscr{F}_n\}$ defined by $\mathscr{F}_n = \sigma(Z_1, Z_2, \cdots, Z_n)$.*

*(ii)* *$\mathbb{E}[\tau_N] = \frac{N}{1-2\epsilon}$.*

*Proof.*

(*i*) We need to verify that $\mathbb{E}[X_{n+1} + (2\epsilon-1)(n+1)|\mathscr{F}_n] = X_n + (2\epsilon-1)n$.

$$\begin{aligned}
&\mathbb{E}[X_{n+1} + (2\epsilon-1)(n+1)|\mathscr{F}_n] \\
&= \mathbb{E}[X_n + Z_{n+1}|\mathscr{F}_n] + (2\epsilon-1)(n+1) \\
&= \mathbb{E}[X_n|\mathscr{F}_n] + \mathbb{E}[Z_{n+1}|\mathscr{F}_n] + (2\epsilon-1)n \\
&= X_n + \mathbb{E}[Z_{n+1}] + (2\epsilon-1)(n+1) \\
&= X_n + (1-2\epsilon) + (2\epsilon-1)(n+1) \\
&= X_n + (2\epsilon-1)n
\end{aligned}$$

(*ii*) Apply the optional stopping theorem (the conditions for OST can be easily checked) to $X_{\tau_N} + (2\epsilon-1)\tau_N$, we get that $\mathbb{E}[X_{\tau_N} + (2\epsilon-1)\tau_N] = \mathbb{E}[X_1 + (2\epsilon-1)] = \mathbb{E}[Z_1] + (2\epsilon - 1) = 0$. So $\mathbb{E}[\tau_N] = \frac{\mathbb{E}[X_{\tau_N}]}{1-2\epsilon} = \frac{N}{1-2\epsilon}$ since $X_{\tau_N} = N$ from the definition of $\tau_N$.

□

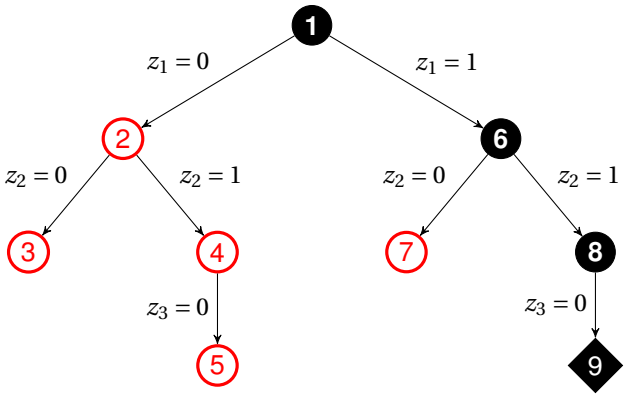



Figure 6: An example search trace by a policy. The solid black nodes ($1 \to 6 \to 8 \to 9$) make up the best trace to a terminal state in retrospect. The empty red nodes are the mistakes made during this search procedure. Every mistake increases the distance to the target node (node 9) by 1 unit, while every correct decision decreases the distance by 1 unit.

Now onto the proof for the Theorem 2.

*Proof.* We consider the search problem as a 1-dimensional random walk (see Figure 6). The random walk starts at the origin and proceeds in an episodic manner. The goal is to reach the point $N$ and at each time step, a wrong decision is equivalent to moving 1 unit to the left whereas a right decision is equivalent to moving 1 unit to the right. The error rate of the policy determines the probabilities of moving left and right. Thus the search problem can be reduced to 1-dimensional random walk, so we can invoke the previous lemma and assert that the expected number of time steps before reaching a feasible solution is $\frac{N}{1-2\epsilon}$. □

This theorem allows us to measure the impact of error rates on the expected number of actions.

**Corollary 3.2.** *With two policies $\pi_1$ and $\pi_2$ with corresponding error rates $0 < \epsilon_1 < \epsilon_2 < \frac{1}{2}$, $\pi_2$ takes $\frac{1-2\epsilon_1}{1-2\epsilon_2}$ times more actions to reach a feasible state in expectation.*

We can further apply Markov's inequality to understand the tail of the distribution on the number of actions. Let $X$ be the random variable representing the number of actions. And we get $\mathbb{P}(X \geq \frac{N^2}{1-2\epsilon}) \leq \frac{1}{N}$. This result indicates that the probability mass of the distribution beyond $\frac{N^2}{1-2\epsilon}$ is small, however, it is still interesting to zoom in on this tail region.

**Theorem 4.** *Let $\mathbb{P}_1$ and $\mathbb{P}_2$ be probability distributions on number of actions for two policies $\pi_1$ and $\pi_2$ with error rates $\epsilon_1$ and $\epsilon_2$. Assume $0 < \epsilon_1 < \epsilon_2 < \frac{1}{2}$. Let $m$ be an integer that is at least $N$ and has the same parity. Then $\frac{\mathbb{P}_2(X=m)}{\mathbb{P}_1(X=m)} = \exp(\frac{\alpha}{2}m - \frac{\beta}{2}N)$ where $\alpha = \log\frac{\epsilon_2(1-\epsilon_2)}{\epsilon_1(1-\epsilon_1)} > 0$ and $\beta = \log\frac{\epsilon_2(1-\epsilon_1)}{\epsilon_1(1-\epsilon_2)}$.*

As a result, the ratio grows exponentially in the number of actions. So even a small improvement on the error rate can make a big difference on the tail probability distribution.

Next we prove Theorem 4.

*Proof.* Let $m \geq N$ and $f(m)$ be the number of possible execution traces that reaches a first feasible solution at the $m$th time step. Assume in the process, the policy made $a$ right choices and $b$ wrong choices, then we have $a + b = m, a - b = N$. So $a = \frac{m+N}{2}, b = \frac{m-N}{2}$. This is the reason we need $m$ and $N$ to have the same parity. Since the probability is $1-\epsilon$ for the policy to make a right choice and $\epsilon$ for a wrong one and the choices are independent of each other, we have that $\mathbb{P}(X = m) = f(m)(1-\epsilon)^{(m+N)/2}\epsilon^{(m-N)/2}$. Substitute into the ratio computation, we have the desired result $\frac{\mathbb{P}_2(X=m)}{\mathbb{P}_1(X=m)} = \exp(\frac{\alpha}{2}m - \frac{\beta}{2}N)$ where $\alpha = \log\frac{\epsilon_2(1-\epsilon_2)}{\epsilon_1(1-\epsilon_1)}$ and $\beta = \log\frac{\epsilon_2(1-\epsilon_1)}{\epsilon_1(1-\epsilon_2)}$. Since we assume that $0 < \epsilon_1 < \epsilon_2 < \frac{1}{2}$, it follows that $\alpha = \log\frac{\epsilon_2(1-\epsilon_2)}{\epsilon_1(1-\epsilon_1)} > 0$. □

## C MILP formulation of risk-aware path planning

This section describes the MILP formulation of risk-aware path planning solved in Section 6. Our formulation is based on the MILP-based path planning originally presented by [Schouwenaars et al., 2001], combined with risk-bounded constrained tightening [Prékopa, 1999]. It is a similar formulation as that of the state-of-the-art risk-aware path planner pSulu [Ono et al., 2013] but without risk allocation.

We consider a path planning problem in $\mathbb{R}$, where a path is represented as a sequence of $N$ way points $x_1, \cdots x_N \in X$. The vehicle is governed by a linear dynamics given by:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k + w_k \\ u_k &\in U, \end{aligned}$$

where $U \subset \mathbb{R}^m$ is a control space, $u_k \in U$ is a control input, $w_k \in \mathbb{R}^n$ is a zero-mean Gaussian-distributed disturbance, and $A$ and $B$ are $n$-by-$n$ and $n$-by-$m$ matices, respectively. Note that the dynamic of the mean and covariance of $x_i$, denoted by $\bar{x}_i$ and $\Sigma_i$, respectively, have a deterministic dynamics:

$$\begin{aligned} \bar{x}_{k+1} &= A\bar{x}_k + Bu_k + w_k \\ \Sigma_{k+1} &= A\Sigma A^T + W, \end{aligned} \tag{1}$$

where $W$ is the covariance of $w_k$. We assume there are $M$ polygonal obstacles in the state space, hence the following linear constraints must be satisfied in order to be safe (as in Figure 7):

$$\bigwedge_{k=1}^{N} \bigwedge_{i=1}^{M} \bigvee_{j=1}^{L_i} h_{ij} x_k \leq g_{ij},$$

where $\wedge$ is conjunction (i.e., AND), $\vee$ is disjunction (i.e., OR), $L_i$ is the number of edges of the $i$-th obstacle, and $h_{ij}$ and $g_{ij}$ are constant vector and scaler, respectively. In order for each of the linear constraints to be satisfied with

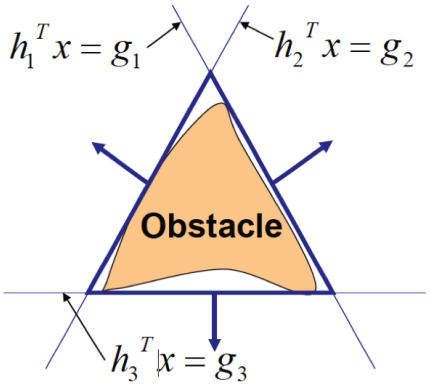



Figure 7: Representation of polygonal obstacle by disjunctive linear constraints

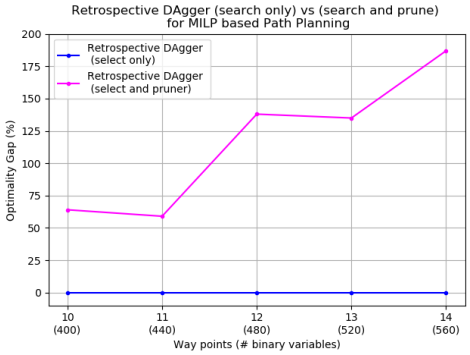



Figure 8: Comparison of optimality gap between Retrospective DAgger (select only) and Retrospective DAgger (select and prune)

the probability of $1-\delta_{kij}$, the following has to be satisfied:

$$\bigwedge_{k=1}^{N}\bigwedge_{i=1}^{M}\bigvee_{j=1}^{L_i} h_{ij}\bar{x}_k \le g_{ij} - \Phi(\delta_{kij}) \tag{2}$$

$$\Phi(\delta_{kij}) = -\sqrt{2h_{ijk}\Sigma_{x,k}h_{ijk}^T}\,\mathrm{erf}^{-1}(2\delta_{ijk}-1),$$

where $\mathrm{erf}^{-1}$ is the inverse error function.

The problem that we solve is, given the initial state $(\bar{x}_0, \Sigma_0)$, to find $u_1 \cdots u_N \in U$ that minimizes a linear objective function and satisfies (1) and (2). An arbitrary nonlinear objective function can be approximated by a piecewise linear function by introducing integer variables. The disjunction in (2) is also replaced by integer variables using the standard Big M method. Therefore, this problem is equivalent to MILP. In the branch-and-bound algorithm, the choice of which linear constraint to be satisfied among the disjunctive constraints in (2) (i.e., which side of the obstacle $x_k$ is) corresponds to which branch to choose at each node.

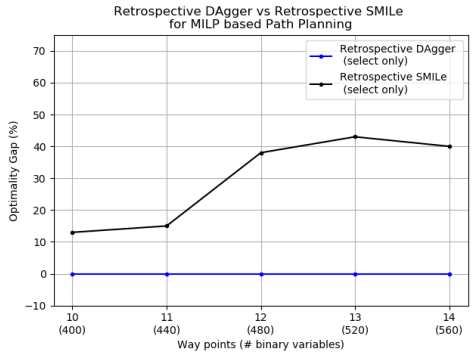



Figure 9: Comparison of optimality gap between Retrospective DAgger (select only) and Retrospective SMILe (select only)

## D Risk-aware Planning Dataset Generation

We generate 150 obstacle maps. Each map contains 10 rectangle obstacles, with the center of each obstacle chosen from a uniform random distribution over the space $0 \le y \le 1$ , $0 \le x \le 1$. The side length of each obstacle was chosen from a uniform distribution in range $[0.01, 0.02]$ and the orientation was chosen from a uniform distribution between 0° and 360°. In order to avoid trivial infeasible maps, any obstacles centered close to the destination are removed.

## E Retrospective DAgger vs Retrospective SMILe for Maze Solving

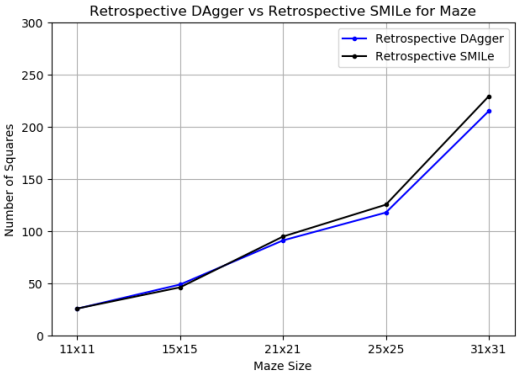



Figure 10: Average explored number of squares for Retrospective DAgger and Retrospective SMILe.

## F Additional Experiments on Risk-aware Planning

In this section, we present a comparison of Retrospective DAgger with two different policy classes for MILP based Path Planning, namely a combination of both select and prune policy as described in [He et al., 2014] against select policy alone. We compare their optimality gap by first

running the Retrospective DAgger (select only) until termination and then limiting the Retrospective Dagger (search and prune) to the same number of explored nodes. Figure 8 depicts a comparison of optimality gap with varying number for waypoints. We observe that Retrospective DAgger (select only) performs much better in comparison to Retrospective DAgger (select and prune).

Next, we present a comparison of Retrospective DAgger (select only) with Retrospective SMILe (select only). We compare the optimality gap by limiting Retrospective SMILe (select only) to the same number of nodes explored by Retrospective DAgger (select only), which is run without any node limits until termination. The results of this experiment are shown in Figure 9. Retrospective DAgger (select only) performs superior to Retrospective SMILe (select only) validating our theoretical understanding of the two algorithms.

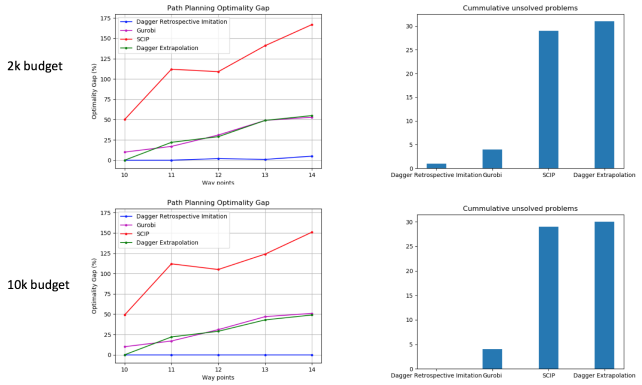



Figure 11: Optimality gap comparisons and number of unsolved problem instances.

Finally, we present statistics on how many instances of MILPs are not solved by each method when given a fixed budget on how many nodes to explore in the branch-and-bound tree. Retrospective DAgger achieves the best record among all the methods compared which implies that it is able to learn a stable and consistent solving policy.

## G Experiments on Combinatorial Auction Test Suite

For completeness of comparison, we evaluate our approach on the same dataset as in He et al. [2014], the Hybrid MILP dataset derived from combinatorial auction problems [Leyton-Brown et al., 2000]. For this experiment, we vary the number of bids, which is approximately the number of integer variables, from 500 to 730. Similar to He et al. [2014], we set the number of goods for all problems to 100 and remove problems that are solved at the root. We use the select and pruner policy class to match the experiments in He et al. [2014] and a similar feature representation to those used in the path planning experiments.

The results of this experiment are shown in Figure 12. We see that neither retrospective imitation learning nor

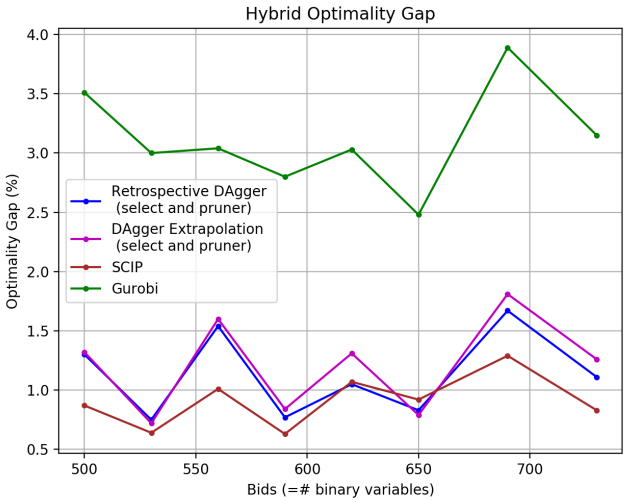



Figure 12: Comparison of optimality gap on the Hybrid combinatorial auction held-out test data.

DAgger Extrapolation (He et al. [2014]) improves over SCIP. Upon further scrutiny of the dataset, we have found several issues with using this combinatorial auction dataset as a benchmark for learning search policies for MILPs and with the evaluation metric in [He et al., 2014].

Firstly, solvers like SCIP and Gurobi are well-tuned to this class of problems; a large proportion of problem instances is solved to near optimality close to the root of the branch-and-bound tree. As shown by Figure 16, Gurobi and SCIP at the root node already achieve similar solution quality as SCIP and Gurobi node-limited by our policy's node counts; hence, exploring more nodes seems to result in little improvement. Thus the actual branch and bound policies matter little as they play a less important role for this class of problems.

Secondly, in this paper, we have chosen to use the number of nodes in a branch-and-bound search tree as a common measure of the speed for various solvers and policies. This is different from that used in [He et al., 2014], where the comparison with SCIP is done with respect to the average runtime. For completeness, we ran experiments using the metric in [He et al., 2014] and we see in Figure 13 that retrospective imitation learning, upon scaling up, achieves higher solution quality than imitation learning and SCIP, both limited by the average runtime taken by the retrospective imitation policy, and Gurobi, limited by average node count.

Instead of using average runtime, which could potentially hide the variance in the hardness across problem instances, using a different limit for each problem instance is a more realistic experiment setting. In particular, the average runtime limit could result in SCIP not being given sufficient runtime for harder problems, leading to SCIP exploring only the root node and a high optimality gap

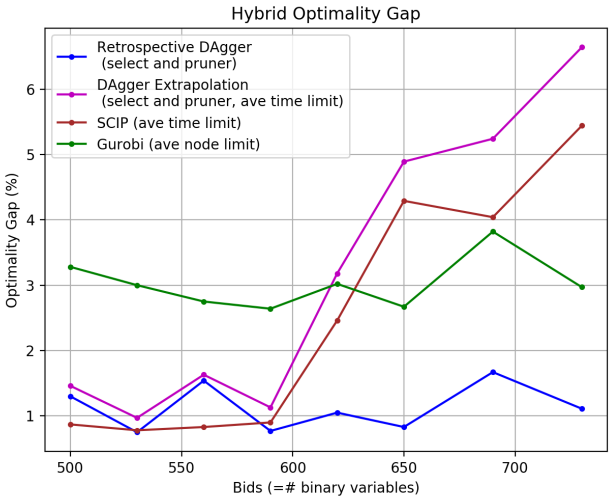



Figure 13: Comparison of optimality gap on the Hybrid combinatorial auction held-out test data using [He et al., 2014] metric.

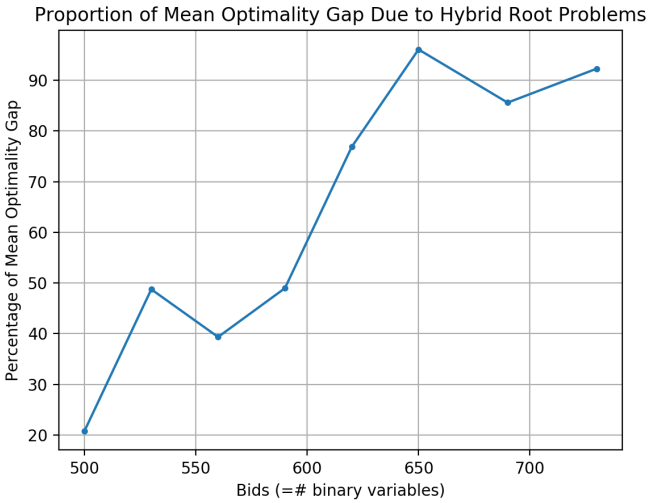



Figure 15: Proportion of mean optimality gap due to Hybrid root problems.

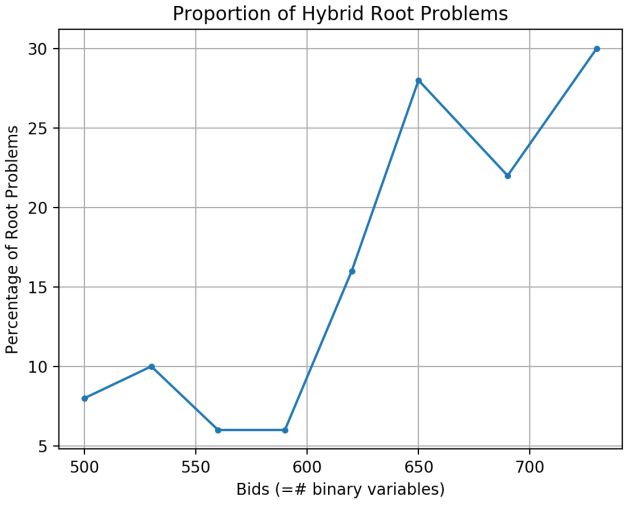



Figure 14: Proportion of Hybrid root problems. Root problems are problems for which SCIP limited by average runtime explores only the root node.

for these problems, which we call "root problems". As Figure 14 shows, a significant proportion of the Hybrid held-out test set is root problems on larger scales. Figure 15 shows that the majority of the mean optimality gap of SCIP limited by average runtime is due to the optimality gap on the root problems in the Hybrid dataset; for larger scale problems, this proportion exceeds 80%, showing that limiting by average runtime heavily disadvantages SCIP.

Another issue is using runtime as the limiting criterion. From our observations, SCIP spends a substantial amount of time performing strong branching at the root to ensure good branching decisions early on. Limiting the runtime results in a limited amount of strong branching; as shown by Figure 17, SCIP limited by the average runtime of our retrospective imitation policies performs significantly less strong branching calls than SCIP limited by node counts, especially at larger problem sizes. In contrast, limiting the number of nodes does not limit the amount of strong branching since strong branching does not contribute to the number of nodes in the final branch-and-bound search tree. Considering the importance of strong branching for SCIP, we feel that only by allowing it can we obtain a fair comparison.

As a result of the above reasons, we decided that the combinatorial auction dataset is not a good candidate for comparing machine learning methods on search heuristics and that the metric used in [He et al., 2014] is not the best choice for validating the efficacy of their method.

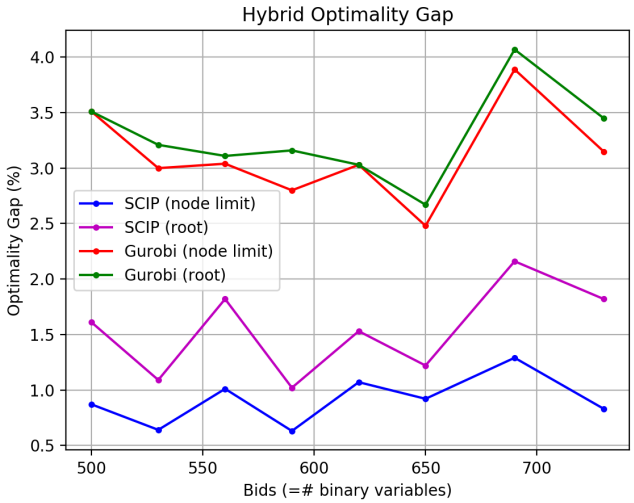



Figure 16: Comparison of optimality gap achieved by SCIP and Gurobi node-limited and at the root on the Hybrid combinatorial auction held-out test data.

## H Exploration Strategy

For retrospective imitation learning to succeed in scaling up to larger problem instances, it is important to enable exploration strategies in the search process. In our experiments, we have found the following two strategies to be most useful.

- $\epsilon$-greedy strategy allows a certain degree of random exploration. This helps learned policies to discover new terminal states and enables retrospective imitation learning to learn from a more diverse goal set. Discovering new terminal states is especially important when scaling up because the learned policies are trained for a smaller problem size; to counter the domain shift when scaling up, we add exploration to enable the learned policies to find better solutions for the new larger problem size.
- Searching for multiple terminal states and choosing the best one as the learning target. This is an extension to the previous point since by comparing multiple terminal states, we can pick out the one that is best for the policy to target, thus improving the efficiency of learning.
- When scaling up, for the first training pass on each problem scale, we collect multiple traces on each data point by injecting 0.05 variance Gaussian noise into the regression model within the policy class, before choosing the best feasible solution.

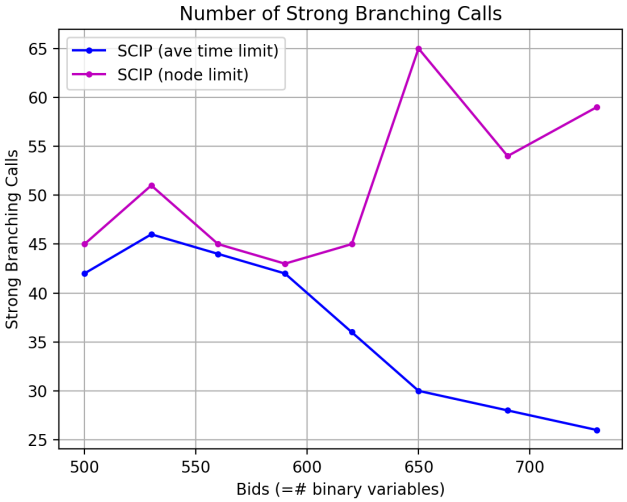



Figure 17: Number of strong branching calls at root for SCIP limited by average runtime and node count at every data point of retrospective imitation learning.